Fundamenta Informaticae 30 (1997) 23–41
IOS Press

Deriving Invariants and Constraints from Action Theories*

Yan Zhang
Department of Computing
University of Western Sydney, Nepean
Kingswood, NSW 2747, Australia
E-mail: yan@st.nepean.uws.edu.au

Norman Y. Foo
Department of Artificial Intelligence
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
E-mail: norman@cse.unsw.edu.au

Abstract. Recent work on reasoning about action has shown that there exists an interesting connection between action specifications and state constraints — it is possible to extract state constraints from action specifications. This work provides us another way to describe the behaviour of dynamic systems. In this paper, we address the problem of generating action invariants from action specifications, and generalizing action invariants into state constraints. We first propose a persistence-based formalism of actions, and show that the generation of action invariants is achieved from action specifications by reasoning about persistence. We then investigate the generalization of action invariants into state constraints. Blocks world examples illustrate the general procedure throughout.

Keywords: knowledge representation, logics for artificial intelligence, reasoning about action

1. Introduction

In dynamic systems (i.e., database systems, planning systems and dynamic physical systems), the system behaviour is governed by actions (events) occurring in the system and constraints. In particular, actions determine the state change of the system, while constraints specify some properties which must be satisfied in all system states. Traditionally, actions and state constraints are independently specified by the system designer.

However, recent work on reasoning about action has shown that actions and constraints have deep connections. For example, Ginsberg and Smith showed that state constraints can

*This research is supported in part by a grant from the Australian Research Council. Most of the research was done while both authers were with the Basser Department of Computer Science at University of Sydney. We would like to thank Fangzhen Lin for detailed discussions on this topic. We also thank the anonymous referee for valuable criticisms.

be derived from a collection of frame axioms within their action formalization [3], while Reiter presented some observations that state constraints may be implied in action axioms in some situations [8, 9]. Furthermore, by introducing induction axioms in the situation calculus [5], Reiter provided a formal method to prove properties of states from action axioms [10]. Other researchers also made similar observations [6, 11].

The work of extracting state constraints from action specifications is important. As illustrated by Reiter, in dynamic systems modeling for purposes such as reasoning about the physical world, planning applications and integrity constraint verifications in databases, it is necessary to prove properties that are true in all world states accessible from the initial state [10], and these properties are exactly the constraints of the system. On the other hand, investigating this problem will reveal that contrary to the normal viewpoint, system constraints are intimately related to the events that are considered to be occurring within the system. Knowing this fact, we can understand why some systems (eg., STRIPS [2]) are workable without specifying constraints explicitly. The other potential use of this work is in hybrid system specifications, where the system may be configured using some constraints and some action specifications. Consistency checks for such hybrid specification require a theory of how actions and constraints interact.

In this paper, we address the problem of generating action invariants and generalizing action invariants into state constraints. Generally, an action invariant expresses a property that is preserved by all actions of that class, while state constraints characterize those states of the world reachable by the actions. The paper is organized as follows. The following section presents the general problem and basic idea of our method. The third section introduces the formal aspect of our approach. A persistence-based formalism of action is described in this section. The fourth section addresses the problem of generating action invariants from action specifications. Based on this work, the fifth section shows how an action invariant can be generalized into a state constraint. Finally, the sixth section presents the conclusion of the paper.

2. The General Problem

To illustrate the connection between action specifications and state constraints, we first consider a blocks world example as follows. Let us consider an action of stacking block a on b, where b is another block or a location. The pre-condition and post-condition of $stack(a, b)$ can be specified as

$$Pre_{stack(a,b)} \equiv block(a) \wedge inhand(a) \wedge clear(b), \tag{1}$$

and

$$Post_{stack(a,b)} \equiv on(a, b). \tag{2}$$

The above pre-condition and post-condition say that the condition of performing action $stack(a, b)$ is that a must be a block, a is held in hand and there is nothing on b, and after performing this action, a is on b.

Intuitively, if the pre-condition of an action is satisfied in the initial state[1], the performance of an action will change the initial state to a resulting state in which the post-condition of the action is satisfied. From our common sense, it is reasonable to assume that the change of a state should only involve those literals in the state that are *relevant* to the action and those *irrelevant* literals of the state should be *persistent* (eg., moving a block should not change the color of the block). Let us call this assumption the *principle of persistence*[2].

[1]We will define the state formally later. At the moment, it is sufficient to consider a state to be an interpretation of a first order language.

[2]We will formalize this principle in section 3.

Suppose S is an initial state in which the pre-condition of $stack(a,b)$ is satisfied, i.e., $S \models Pre_{stack(a,b)}$, and S' is the resulting state after performing $stack(a,b)$ on S. Based on the principle of persistence described above, obviously, we should have the following relationships:

$\{\neg on(a,b), block(a), inhand(a), clear(b)\} \subseteq S$, and
$\{on(a,b), block(a), \neg inhand(a), \neg clear(b)\} \subseteq S'$.

Note that subset $\{\neg on(a,b), inhand(a), clear(b)\}$ of S is the only part of S that is affected by action $stack(a,b)$, while $S - \{\neg on(a,b), inhand(a), clear(b)\}$ is the *persistent set* of S with respect to action $stack(a,b)$. From the above relationships, it is not difficult to observe that formulas

$$on(a,b) \supset block(a), \tag{3}$$

$$on(a,b) \supset \neg inhand(a), \text{and} \tag{4}$$

$$on(a,b) \supset \neg clear(b) \tag{5}$$

are satisfied in both the initial state S and the resulting state S'. For instance, consider formula (3). As $on(a,b)$ is not satisfied in S, (3) is then satisfied in S. On the other hand, after performing action $stack(a,b)$, $on(a,b)$ becomes true in the resulting state S'. Since $block(a)$ is ture in both S and S', together with the fact that $on(a,b) \in S'$, it implies that formula (3) is also satisfied in S'. Similar justifications can be made for (4) and (5). Therefore, (3), (4) and (5) can be viewed *invariants* of action $stack(a,b)$ in this case. Further, if there are only two objects a and b in this blocks world, formulas (3), (4) and (5) can be treated as the state constraints that characterize those *reasonable* states of this blocks world.

An interesting point here is that constraints (3), (4) and (5) seem only related to the pre-condition and post-condition of $stack(a,b)$. A natural question from such observation is: Is there a general syntactic method to derive state constraints from the action specifications?

Our task in this paper is then to formalize the intuitive procedure described above. The basic idea of our approach can be described as follows. Basicly, we do this in two steps: first, for any *specific* action, we derive an invariant of this action from its pre- and post-conditions; second, we generalize the invariant of the specific action to the invariant of the *generic* action, and then investigate the conditions under which a generic action invariant can act as a state constraint in the system.

Note that here we mention specific actions and generic actions. In our example presented above, action $stack(a,b)$ is a specific action which means stacking block a on b, while a generic action is an abstraction of a class of specific actions which have the same effects but are performed on different objects. For example, we can define a generic action $Stack(x,y)$ specifying that stacking block x on y. We can replace variables x and y with different object constants obtaining a class of specific actions, eg., $\{stack(a,b)$, $stack(c,d), \cdots\}$. Thus, we finally hope to generalize (3) — (5) to formulas like

$$on(x,y) \supset block(x), \tag{6}$$

$$on(x,y) \supset \neg inhand(x), \text{and} \tag{7}$$

$$on(x,y) \supset \neg clear(y), \tag{8}$$

such that they are satisfied in any world state reachable by any specific action in $\{stack(a,b)$, $stack(c,d), \cdots\}$. In this sense, formulas (6) — (8) act as state constraints in our blocks world described above.

3. Effects of Actions

In this section, we introduce a persistence-based formalism of actions. Consider a first order language with equality $\mathcal{L}$. We use an interpretation of language $\mathcal{L}$ to represent the *state of the world*. A state of the world may be changed to another state by performing some action. An *action description* A is defined by pre-condition Pre_A and post-condition $Post_A$, where Pre_A and $Post_A$ are first order formulas. The free variables in these formulas are to be interpreted as action parameters, while the bound variables are connected with tests on states. An action description therefore defines a class of specific actions, denoted as $A = \{act_1, act_2, \cdots\}$, where for each $act \in A$, its pre-condition and post-condition are instances of Pre_A and $Post_A$ respectively. We denote act's pre-condition and post-condition as $Pre_{A(act)}$ and $Post_{A(act)}$ respectively. In the rest of the paper, we also use Pre_{act} and $Post_{act}$ to denote the pre-condition and post-condition of act respectively, if act's action description is not important in our discussion.

To simplify our presentation, we will assume that if two (or more) different interpretations I_1 and I_2 are involved in our discussion, then $|I_1| = |I_2|$, i.e., the universe of I_1 is the same as the universe of I_2. Note that this assumption does not mean that we restrict our language to a specific universe. It just means that we are discussing different interpretations with the same universe. Usually, $|I_1| = |I_2|$ is required in a model-based semantic analysis in which two different interpretations (models) are involved. For instance, in Lifschitz's definition for the model semantics of circumscription, the condition $|I_1| = |I_2|$ is explicitly stated [4].

Example 3.1. The blocks world. We define an action description of $Stack$ as follows[3]

$$Pre_{Stack} \equiv block(x) \wedge \forall z.\neg on(x,z) \wedge \forall v.\neg on(v,y) \wedge \exists w.(on(y,w) \vee loc(y)), \tag{9}$$

$$Post_{Stack} \equiv on(x,y). \tag{10}$$

The action description above defines a *generic action* of stacking block x on y[4]. The pre-condition of this action says that x is in hand ($block(x) \wedge \forall z.\neg on(x,z)$), y is clear ($\forall v.\neg on(v,y)$) and y is located on somewhere or y itself is a location ($\exists w.(on(y,w) \vee loc(y))$). The post-condition simply says that after performing this action, block x is on y. A specific action $stack(a,b)$ of stacking block a on b is then specified as follows:

$$Pre_{stack(a,b)} \equiv block(a) \wedge \forall z.\neg on(a,z) \wedge \forall v.\neg on(v,b) \wedge \exists w.(on(b,w) \vee loc(b)), \tag{11}$$

$$Post_{stack(a,b)} \equiv on(a,b), \tag{12}$$

where a and b are constants of $\mathcal{L}$. $\Box$

We now consider how to specify the effect of an action in the resulting state after performing the action. We say that an action act is *applicable* to a state S if $S \models Pre_{act}$. For the resulting state S', it is clear that we require $S' \models Post_{act}$. However, this is not enough because there may be some states satisfying such a condition but violating the persistence requirement for other facts. We will define a precise representation for the possible resulting states based on the *persistence principle*. Generally, the persistence principle says that a fact in the initial state persists during the performance of an action, if this fact is logically irrelevant to those facts that must change or may change with respect to the action.

Definition 3.1. Suppose action act is applicable to the state S_0. We define the *persistent set* $\Delta(S_0, act)$ of S_0 with respect to action act as follows:

---

[3] Note that we use an alternative definition to specify $Stack$ compared with the one described in section 2.

[4] In the rest of this paper, we will alternately use the terminologies *generic action* and *action description* for convenience.

$\Delta_1 = \{f \mid Post_{act} \models \neg f, \text{ where } f \in S_0\}$,

$\Delta_2 = \{f_1, \cdots, f_k \mid Post_{act} \models \bigvee_{i=1}^{k}[\neg]f_i, \text{ where } \{f_1, \cdots, f_k\} \subseteq S_0 \text{ and for any } M \subset N = \{1, \cdots, k\},\ Post_{act} \not\models \bigvee_{j\in M}[\neg]f_j\}$,

$\Delta(S_0, act) = S_0 - \Delta_1 - \Delta_2$.

The notation $[\neg]$ means that the negation sign $\neg$ may or may not appear.

The explanation of the above definition is as follows. First of all, we say that a fact (i..e., a literal) f in S_0 is *non-persistent* with respect to action act, if f is inconsistent with $Post_{act}$. In other words, f's truth value must be changed after performing action act. Second, we say that facts $f_1, \cdots, f_k$ in S_0 are *indefinite*, if $Post_{act}$ non-trivially entails the disjunction $\bigvee_{i=1}^{k}[\neg]f_i$. It means that after performing action act, for each f_i, its truth value is indefinite in the resulting state. In other words, f_i may or may not change, but we do not know which is the case. Finally, we define that a fact in S_0 is *persistent* if it is neither non-persistent nor indefinite with respect to action act.

Note that in the above definition, we only consider the indefiniteness of the disjunction entailed by $Post_{act}$, and ignore the other case of indefiniteness entailed by $Post_{act}$, i.e., $Post_{act} \models \exists x.P(x)$. Intuitively, if $Post_{act} \models \exists x.P(x)$, while such entailment is non-trivial, i.e., $Post_{act} \not\models p(c)$ for any constant c, every instance of $\exists x.p(x)$ has a possibility to be true in the resulting state but we do not know which one. In order to reflect this semantics in the definition above, we need the domain closure assumption [7]. Thus, whenever there exists the case of $Post_{act} \models \exists x.P(x)$, we assume $\exists x.p(x) \equiv (p(c_1 \vee \cdots \vee p(c_n))$, i.e., every instance of $\exists x.p(x)$ is indefinite with respect to the action.

We can then define the possible resulting state as follows.

Definition 3.2. Let action act be applicable to the state S_0. We say that state S is a *possible state* resulting from the performance of action act on S_0, if $S \models Post_{act}$ and $\Delta(S_0, act) \subseteq S$. We denote the set of all possible resulting states as $Result(S_0, act)$.

Note that in the above descriptions, we do not consider state constraints. We assume that action specifications *completely* specify the effects of actions. Based on this assumption, we will show that state constraints can be extracted from action specifications.

One advantage of our method is that it can properly represent *indefinite effects* of actions (eg., the post-condition of an action may be a disjunction), while most of other approaches usually have difficulties in dealing with this problem. For example, consider the action of tossing a coin with the pre-condition $heads$ and post-condition $heads \vee tails$. Using our method described above, we have two possible resulting states after the action of tossing a coin: one with the outcome $heads$ and the other with the outcome $tails$. Obviously, this solution is desired. Using other approaches, on the other hand, such as the PWA [3] or PMA [12], only one possible outcome $heads$ is obtained, and this result is obviously implausible.

The above action formalism is a simplified version of our persistence theory of actions proposed in [13], where state constraints were explicitly taken into account. In fact, the persistence idea has wide applications in dynamic systems modeling. For instance, it can be used in temporal reasoning and knowledge base updates. More details of its applications in these areas can be found in [14, 15].

4. Generating Action Invariants

4.1. Preliminaries

In this section, we propose a formal method to derive invariants of actions. We first introduce the following concepts. A formula F is *pure equational* if the only predicate symbol occurring in F is the equality symbol $=$. For example, $x = a \vee y = b$ is a pure equational formula. F is *most-simplified* if there does not exist another formula F' such that $F \equiv F'$ and F' is an *occurrence* in F but F is *not* an occurrence in F'. For instance, let F be $(p \wedge q) \vee p$. F is not most-simplified since $F \equiv p$ and p appears in F but F does not appear in p. To exclude the trivial case in our discussions in the rest of the paper, we will only consider the satisfiable, non-pure equational and most-simplified formulas in the following presentation.

Definition 4.1. A formula F is called *definite* if for any two interpretations I_1, I_2 such that $I_1 \models F$ and $I_2 \models F$, and for each non-equality predicate symbol p appearing in F, if $p(x,t)$ occurs in F, where x and t are tuples of variables and constants respectively (x or t may be empty), then $I_1 \models p(t',t)$ iff $I_2 \models p(t',t)$ for any tuple of constants t' that has the same arity as x. F is *indefinite* if F is neither definite nor a tautology.

Intuitively, a formula F is definite if the truth value of each predicate occurring in F is definite in any interpretation in which F is satisfied. For example, formulas $\forall x.p(x), \forall x.q(x,c)$ and $p(c_1) \wedge q(c_2,c_3)$ are definite, while formulas $\exists x.p(x)$ and $p(c_1) \vee q(c_2,c_3)$ are indefinite.

Definition 4.2. Let F be a formula and I an interpretation that satisfies F, i.e., $I \models F$. We define a subset of I to be an *extension* of F, denoted by $E(F)_I$, as follows. For each non-equality predicate symbol p occurring in F, if $p(x,t)$ appears in F where x and t are tuples of variables and constants respectively (x or t may be empty), then for any tuple t' of constants that has the same arity as x, $[\neg]p(t',t)$ is in $E(F)_I$ iff $[\neg]p(t',t)$ is in I.

The intention of the extension of formula F is to capture the set of ground literals in I that represent interpretations of all non-equality predicates occurring in F. Obviously, an extension of a formula is interpretation-dependent, and a formula may have multiple extensions. For example, consider any interpretation I of $\mathcal{L}$, where $|I| = \{c_1, \cdots, c_n, \cdots\}$. Formula $p(c)$ has unique extension of $\{p(c)\}$, and for formula $\forall x.p(x,c)$, its unique extension is $\{p(c_1,c), \cdots, p(c_n,c), \cdots\}$ where c is some c_i in $|I|$. On the other hand, formula $p(c_1) \vee q(c_2)$ has three extensions $\{p(c_1), \neg q(c_2)\}$, $\{\neg p(c_1), q(c_2)\}$ and $\{p(c_1), q(c_2)\}$, while formula $\exists x.p(x)$ has an infinite number of extensions:

$$
\begin{array}{l}
\{p(c_1), \neg p(c_2), \cdots, \neg p(c_n), \cdots\},\\
\{\neg p(c_1), p(c_2), \neg p(c_3), \cdots, \neg p(c_n), \cdots\},\\
\cdots,\\
\{\neg p(c_1), \neg p(c_2), \cdots, \neg p(c_{n-1}), p(c_n), \cdots\},\\
\cdots,\\
\{p(c_1), p(c_2), \neg p(c_3), \cdots, \neg p(c_n), \cdots\},\\
\cdots,\\
\{p(c_1), \cdots, p(c_{n-1}), \neg p(c_n), \cdots\},\\
\{p(c_1), \cdots, p(c_{n-1}), p(c_n), \cdots\},\\
\cdots.
\end{array}
$$

The following results present some important properties of extensions of formulas.

Theorem 4.1. Let F be a formula and $E(F)$ an extension of F. $E(F)$ is unique iff F is definite.

Proof:
Let I, I' be any two interpretations such that $I \models F$ and $I' \models F$. Consider $E(F)_I$ and $E(F)_{I'}$. According to Definition 4.2, for each non-equality predicate symbol p occurring in F, $[\neg]p(t,'t) \in E(F)_I$ iff $I \models [\neg]p(t,'t)$, and $[\neg]p(t,'t) \in E(F)_{I'}$ iff $I' \models [\neg]p(t,'t)$, where $p(x,t)$ appears in F, x and t are tuples of variables and constants respectively, and t' is any tuple of constants that has the same arity as x. It follows that $E(F)_I = E(F)_{I'}$ if and only if $I \models [\neg]p(t,'t)$ iff $I' \models [\neg]p(t,'t)$. From the definition of definite formula (Definition 4.1), this means that $E(F)_I = E(F)_{I'}$ iff F is definite. $\Box$

Theorem 4.2. Let F_1, F_2 be two formulas, and I an interpretation of $\mathcal{L}$ such that $I \models F_1 \wedge F_2$. Then $E(F_1 \wedge F_2)_I = E(F_1)_I \cup E(F_2)_I$.

Proof:
According to Definition 4.2, $E(F_1 \wedge F_2)_I = \{[\neg]p(t', t) \mid [\neg]p(t', t) \in I$, where $p(x, t)$ occurs in F_1 or $F_2\} = \{[\neg]p(t', t) \mid [\neg]p(t', t) \in I$, where $p(x, t)$ occurs in $F_1\} \cup \{[\neg]p(t', t) \mid [\neg]p(t', t) \in I$, where $p(x, t)$ occurs in $F_2\} = E(F_1)_I \cup E(F_2)_I$. $\Box$

In order to derive an invariant of an action from its specification (pre-condition and post-condition), we also need to consider the persistent part of its pre-condition. Let $E(Pre_{act})_S$ be an *extension* of Pre_{act} in S where act is applicable to S. According to Definition 4.2, Pre_{act} specifies a subset of literals in state S (i.e., $E(Pre_{act})_S \subseteq S$).

Clearly, if the execution of an action does not affect its pre-condition, then its pre-condition will still be satisfied in the resulting state, that is, the pre-condition itself is an invariant (we will give the formal definition of an action invariant below). However, in most cases, the action may affect some fact(s) specified by its pre-condition, while other facts specified by the pre-condition remain unchanged. For example, in the definition of action $stack(a, b)$ described in Example 3.1, only the truth value of $on(a, b)$ will be changed from false to true after performing $stack(a, b)$ while other facts specified by $Pre_{stack(a,b)}$ remain unchanged (eg., $block(a)$ and $\exists w.(on(y, w) \vee loc(y))$). Now, the persistent part of the initial state related to the pre-condition of an action is formally defined as follows.

Definition 4.3. Suppose action act is applicable to state S. We call

$$P(Persist|act)_S = E(Pre_{act})_S \cap \Delta(S, act) \tag{13}$$

the *persistent projection* of act with respect to state S.

Example 4.1. The blocks world (continued). Consider a blocks world in which there are only three objects block a, location b and location c, and the only action occurring in this blocks world is $stack(a, b)$ which is defined by (11) and (12) in Example 3.1. Suppose the initial state of the blocks world is

$$\begin{aligned} S = \{&block(a), loc(b), loc(c),\\ &\neg loc(a), \neg block(b), \neg block(c),\\ &\neg on(a, a), \neg on(a, b), \neg on(a, c)\\ &\neg on(b, a), \neg on(b, b), \neg on(b, c)\\ &\neg on(c, a), \neg on(c, b), \neg on(c, c)\}. \end{aligned}$$

Obviously, $stack(a, b)$ is applicable to S, i.e., $S \models Pre_{stack(a,b)}$. From Definition 4.2, the extension of $Pre_{stack(a,b)}$ in S is

$$E(Pre_{stack(a,b)})_S = S - \{\neg loc(a), \neg block(b), loc(c), \neg block(c), \neg on(c, a), \neg on(c, c)\}.$$

Thus, the persistent projection of $stack(a, b)$ with respect to S is

$$\begin{aligned} P(Persist|stack(a, b)_S &= E(Pre_{stack(a,b)})_S \cap \Delta(S, stack(a, b)\\ &= E(Pre_{stack(a,b)})_S - \{\neg on(a, b)\}. \ \Box \end{aligned}$$

The following result will be useful in our derivation.

Theorem 4.3. If Pre_{act} is definite, then $P(Persist|act)$ is unique, that is, for any S and S', $S \models Pre_{act}$ and $S' \models Pre_{act}$ implies $P(Persist|act)_S = P(Persist|act)_{S'}$.

Proof:
According to Definition 4.3, for any S such that $S \models Pre_{act}$, $P(Persist|act)_S = E(Pre_{act})_S \cap \Delta(S, act)$. From Theorem 4.1, Pre_{act} is definite iff $E(Pre_{act})$ is unique. Thus, $P(Persist|act)_S = E(Pre_{act}) \cap \Delta(S, act)$. On the other hand, since $E(Pre_{act}) \cap \Delta(S, act) \subseteq E(Pre_{act})$ for any S such that $S \models Pre_{act}$, it follows that $E(Pre_{act}) \cap \Delta(S, act)$ is unique, so $P(Persist|act)$ is unique. $\Box$

4.2. Expressibility of Persistent Projection

Now we turn to consider the expressibility problem of the persistent projection of an action. We first define the expressibility of a set of ground literals.

Definition 4.4. Let Δ be a non-empty consistent set of ground literals. We say that Δ is *expressible* by a formula Exp, if and only if for any interpretation I such that $\Delta \subseteq I$, $I \models Exp$.

Note that from the above definition, if Δ is expressed by formula Exp, then for any formula F, Δ is also expressed by $Exp \vee F$. However, not all formulas expressing Δ are of equal interest. For example, if $\Delta = \{p(c_1), q(c_2)\}$, then both formulas $p(c_1) \wedge q(c_2)$ and $(p(c_1) \wedge q(c_2)) \vee p(c_3)$ express Δ though we may be only interested in the first one. Specifically, if $\Delta = \{p(r) \mid r \in |I| \text{ but } r \neq c\}$, then Δ can be expressed by formula $\forall x.(x \neq c \supset p(x))$. Generally, we have the following result.

Theorem 4.4. Let act be an action with pre-condition $\bigwedge_{i=1}^{m} \forall x_i.[\neg]p_i(x_i, t_i)$, where x_i and t_i are tuples of variables and constants respectively (t_i may be empty). Suppose that for each i, p_i changes its values *definitely* at points $t_{i1}, \cdots, t_{ik_j}$ after performing action act[5]. Then the persistent projection $P(Persist|act)$ [6] can be expressed by the formula $Exp_{act} \equiv \bigwedge_{i=1}^{m} \phi_i$, where $\phi_i \equiv \forall x_i.(x_i \neq t_{i1} \wedge \cdots \wedge x_i \neq t_{ik_j} \supset [\neg]p_i(x_i, t_i))$.

Proof:
Case 1. To prove this theorem, we first prove Result 1: if $E = E_1 \cup E_2$, where E_1 and E_2 are sets of ground literals, and E_1 is expressed by F_1 and E_2 is expressed by F_2, then E is expressed by $F_1 \wedge F_2$.

Obviously, for any interpretation I such that $E \subseteq I$, $E_1 \subseteq I$ and $E_2 \subseteq I$. Since E_1 and E_2 are expressed by F_1 and F_2 respectively, from Definition 4.4, $I \models F_1$ and $I \models F_2$, and from this follows $I \models F_1 \wedge F_2$. So, E is expressed by $F_1 \wedge F_2$.

Case 2. We use Result 1 to prove Theorem 4.4. It is enough to prove that for an action act with pre-condition $\forall x.p(x, t)$, if p changes its values definitely at point $t_1, \cdots, t_k$ after performing act, then $P(Persist|act)$ is expressed by formula Exp, where $Exp \equiv \forall x.(x \neq t_1 \wedge \cdots \wedge x \neq t_k \supset p(x, t))$.

Consider $P(Persist|act)$[7]. Obviously, $P(Persist|act) = \{p(t', t) \mid t' \neq t_1 \wedge \cdots \wedge t' \neq t_k$, where $t', t, t_1, \cdots, t_k \in |I|\}$. Trivially, for any interpretation I such that $P(Persist|act) \subseteq I$, $I \models Exp$. □

The above theorem says that if an action act's pre-condition is definite and act only has definite effects, then $P(Persist|act)$ can be always expressed by a formula. However, if pre-condition Pre_{act} is indefinite, then its persistent projection $P(Persist|act)_S$ is not unique, and it is generally infeasible to derive a unified formula to express $P(Persist|act)_S$. But there exist some special cases. Suppose Pre_{act} is indefinite. First, if $E(Pre_{act})_S \subseteq \Delta(S, act)$ for any S to which act is applicable, then every fact in $E(Pre_{act})_S$ is persistent and $P(Persist|act)_S = E(Pre_{act})_S$ is expressed by Pre_{act} itself. Second, if $P(Persist|act)_S = E(Pre_{act})_S \cap \Delta(S, act) = \emptyset$ for any S to which act is applicable, then every fact in $E(Pre_{act})_S$ is non-persistent or indefinite, and no formula exists to express $P(Persist|act)_S$. Generalizing these observations, we have the following theorem.

Theorem 4.5. Let act be an action with pre-condition $F \wedge G \wedge H$ where (i) F is definite, (ii) G is indefinite and $E(G)_S \subseteq \Delta(S, act)$ for any S to which act is applicable, and (iii) H is indefinite and $E(H)_S \cap \Delta(S, act) = \emptyset$ for any S to which act is applicable. Then:

[5] This means that there is no indefinite change for p_i's truth value.

[6] Note that $\bigwedge_{i=1}^{m} \forall x_i.[\neg]p_i(x_i, t_i)$ is definite. It follows that $P(Persist|act)$ is unique according to Theorem 4.1.

[7] Recall that from Theorem 4.3, since Pre_{act} is definite, $P(Persist|act)$ is unique.

1. $P(Persist|act)_S = (E(F) \cap \Delta(S, act)) \cup E(G)_S$[8];
2. If $E(F) \cap \Delta(S, act)$ is expressed by the formula F^*, then $P(Persist|act)_S$ can be expressed by formula $Exp_{act} \equiv F^* \wedge G$.

Proof:
Case 1. From Definition 4.2 and Theorem 4.2, we have:
$P(Persist|act)_S = E(F \wedge G \wedge H)_S \cap \Delta(S, act)$
$= (E(F)_S \cup E(G)_S \cup E(H)_S) \cap \Delta(S, act)$
$= (E(F)_S \cap \Delta(S, act)) \cup (E(G)_S \cap \Delta(S, act)) \cup (E(H)_S \cap \Delta(S, act))$.
From conditions (ii) and (iii), we have $P(Persist|act)_S = (E(F)_S \cap \Delta(S, act)) \cup E(G)_S$. Since F is definite, it follows that $E(F)_S$ is unique for any S such that $S \models F$. So, $P(Persist|act)_S = (E(F) \cap \Delta(S, act)) \cup E(G)_S$
Case 2. Clearly, $E(G)_S$ is expressed by G itself. Then the result directly follows from Result 1 in the proof of Theorem 4.4. □

4.3. Action Invariants

Informally an action invariant presents a property that is preserved by the performance of the action. Now we give the formal definition as follows.

Definition 4.5. A non-tautological formula $Inv(act)$ is an *invariant* of the action act, if for any state S to which act is applicable, i.e., $S \models Pre_{act}$, $S \models Inv(act)$ and $S' \models Inv(act)$, where $S' \in Result(S, act)$.

Obviously, the invariant of an action may be not unique. To reveal the relationship between action specifications and action invariants, we are interested in extracting invariants from pre- and post-conditions of actions. Based on the previous results of expressibility, we finally have the following theorem.

Theorem 4.6. If for any state S where action act is applicable, the persistent projection $P(Persist|act)_S$ of act with respect to S can be expressed by formula Exp_{act}, then formula $Post_{act} \supset Exp_{act}$ is an invariant of act.

Proof:
We need to prove that $S \models Post_{act} \supset Exp_{act}$, and $S' \models Post_{act} \supset Exp_{act}$, where $S \models Pre_{act}$ and $S' \in Result(S, act)$. Since $P(Persist|act)_S = E(Pre_{act})_S \cap \Delta(S, act) \subseteq S$, according to expressibility definition Definition 4.4, $S \models Exp_{act}$, which follows that $S \models Post_{act} \supset Exp_{act}$. Now consider $S' \in Result(S, act)$. Since $S' \models Post_{act}$, we need to prove $S' \models Exp_{act}$. Since $P(Persist|act)_S \subseteq \Delta(S, act) \subseteq S'$, it follows $S' \models Exp_{act}$. □

Example 4.2. The blocks world (continued). Consider action $stack(a, b)$ specified in Example 3.1. From formula (11), according to Theorems 4.1 and 4.5, we know that for any state S such that $S \models Pre_{stack(a,b)}$, we have

$$\begin{aligned} P(Persist|stack(a, b))_S = \{&block(c)\} \cup \\ &\{\neg on(a, r) \mid r \in |S| \text{ but } r \neq b\} \cup \\ &\{\neg on(r', b) \mid r' \in |S| \text{ but } r' \neq a\} \cup \\ &E(\exists w.(on(b, w) \vee loc(b)))_S. \end{aligned}$$

From Theorem 4.4, the set

$$\{block(c)\} \cup \{\neg on(a, r) \mid r \in |S| \text{ but } r \neq b\} \cup \{\neg on(r', b) \mid r' \in |S| \text{ but } r' \neq a\}$$

is expressed by formula

$$F^* \equiv block(a) \wedge \forall z.(z \neq b \supset \neg on(a, z)) \wedge \forall v.(v \neq a \supset \neg on(v, b)).$$

[8]Since F is definite, $E(F)$ is unique from Theorem 4.1.

On the other hand, since $Post_{stack(a,b)} \equiv on(a,b)$ and $on(a,b)$ is irrelevant to $\exists w.(on(b,w) \vee loc(b))$, from Definition 3.1, we have $E(\exists w.(on(b,w) \vee loc(b)))_S \subseteq \Delta(S, stack(a,b))$. Again from Theorem 4.5, it follows that $P(Persist|stack(a,b))_S$ is finally expressed by formula $Exp_{stack(a,b)}$:

$$block(a) \wedge \forall z.(z \neq b \supset \neg on(a,z)) \wedge \forall v.(v \neq a \supset \neg on(v,b)) \wedge \exists w.(on(b,w) \vee loc(b)). \quad (14)$$

So, $Post_{stack(a,b)} \supset Exp_{stack(a,b)}$ is an invariant of action $stack(a,b)$. Simplifying $Post_{stack(a,b)} \supset Exp_{stack(a,b)}$, we finally have the following invariants of action $stack(a,b)$:

$$on(a,b) \supset block(a), \quad (15)$$

$$\forall z.\neg(on(a,b) \wedge z \neq b \wedge on(a,z)), \quad (16)$$

$$\forall v.\neg(on(a,b) \wedge v \neq a \wedge on(v,b)), \quad (17)$$

$$on(a,b) \wedge \neg loc(b) \supset \exists w.on(b,w). \quad (18)$$

The meaning of (15) is obvious. (16) says that if a is on b, then a cannot be somewhere else, while (17) says that if a is on b, then there does not exist other blocks on b. (18) expresses that if block a is on b and b is not a location, then b must be located somewhere. □

Although our action theory described in section 3 properly solves the problem of indefinite effects of actions, there is, however, a restriction with our method of deriving invariants from action specifications. As described above, our derivation of action invariants is based on the expressibility of persistent projection of actions. For an action act, if its persistent projection $P(Persist|act)_S$ is not unique, then it cannot be expressed by a unified formula independently from S, and we cannot derive an invariant of act as the form $Post_{act} \supset Exp_{act}$. Following this observation, we know that if Pre_{act} is *indefinite* and it does not satisfy the properties stated in Theorem 4.5, or Pre_{act} is definite but act has indefinite effects on facts related to Pre_{act} (i.e., Pre_{act} does not satisfy the condition stated in Theorem 4.4), then our method can not extract invariants for this action. In other words, our method only provides a partial solution for deriving invariants from action specifications. But we have noted that the solutions provided here (i.e., Theorems 4.4 and 4.5) are broad enough to cover most situations of action specifications.

5. From Action Invariants to State Constraints

In the previous section, we have discussed the generation of action invariants. In this section, we address the problem of generalizing action invariants into state constraints.

5.1. Generalized Invariants

We first consider invariants of action descriptions. Recall that in our formalism, we define an *action description* to specify a class of actions, in which each action has the "same effect" but is performed on different objects. Particularly, specifying A to be an action description by given Pre_A and $Post_A$, we obtain a class of specific actions $act_1, act_2, \cdots$ by substituting different free variables occurring in Pre_A and $Post_A$ with different constants in the language $\mathcal{L}$. Let $A = \{act_1, act_2, \cdots\}$. So, for any specific action act in A, if $Inv(act)$ is an invariant of act generated from its specification as presented previously, then for *any other* specific action act' in A, there exists an invariant $Inv(act')$ of act', which has the same form as $Inv(act)$ but with different *action parameters*. This is because the generation of action invariants is independent of action parameters. For example, for action $stack(a,b)$, there is an invariant generated from its specification: $on(a,b) \supset block(a)$. For action $stack(c,d)$, on the other hand, formula $on(c,d) \supset block(c)$ is an invariant of $stack(c,d)$.

Definition 5.1. Let $A = \{act_1, act_2, \cdots\}$. For any $act \in A$, suppose that $Inv(act)$ is an invariant of *act* generated from its specification as presented previously. We call the formula $Inv(A)$, which is obtained by replacing different constants in $Inv(act)$ with different universally quantified variables, an *invariant* of A.

For instance, from the specification of action $stack(a,b)$, we generated the invariant $on(a,b) \supset block(a)$, from which we obtain the generalized invariant $on(x,y) \supset block(x)$ for the *generic action Stack*, where variables x and y are universally quantified.

5.2. Relative State Constraints

Now we consider how a generalized action invariant can act as a state constraint. Informally, a *state constraint* is a formula that should be satisfied by any state reachable by *every* action. In this sense, all state constraints characterize the properties of the world where actions occur.

One of the key arguments of this paper is that in some circumstances, action invariants can be generalized to state constraints of the system. To show this, we first introduce the concept of *reachable states* of generic actions.

First of all, we assume that there are only two generic actions A_1 and A_2 in our problem domain. Suppose $Inv(A_1)$ and $Inv(A_2)$ are two invariants of A_1 and A_2 respectively and consider a specific state S_0 such that $S_0 \models Inv(A_1) \wedge Inv(A_2)$. Trivially, we say S_0 is a S_0-*relative reachable state* of A_1 and A_2. Recursively, if S is a S_0-relative reachable state of A_1 and A_2, and there exists some *act* in $A_1 \cup A_2$ such that *act* is applicable to S, i.e., $S \models Pre_{act}$, then for every $S' \in Result(S, act)$, we say S' is a S_0-relative reachable state of A_1 and A_2. The set of all S_0-relative reachable states of A_1 and A_2 is denoted as $R[A_1, A_2]_{S_0}$.

Let us examine the class $R[A_1, A_2]_{S_0}$ a little more closely. It is clear that the cardinality of $R[A_1, A_2]_{S_0}$ depends on the choice of S_0. If we choose a proper starting state S_0, we may get a large class of S_0-relative reachable states of A_1 and A_2.

However, it is possible that we may choose a specific starting state that leads to a small set of S_0-relative reachable states of A_1 and A_2. A special case is when S_0 satisfies $Inv(A_1)$ and $Inv(A_2)$, but every *act* in A_1 and A_2 is not applicable to S_0. In this case, $R[A_1, A_2]_{S_0} = \{S_0\}$. For example, consider the blocks world presented previously, where *Stack* is the only generic action. Suppose that we choose S_0 to be the state in which every block is already located somewhere. Then any specific action in *Stack*, say $stack(a,b)$, is not applicable to S_0 as $S_0 \models \exists z.on(a,z)$ and this results in $S_0 \not\models Pre_{stack(a,b)}$.

$R[A_1, A_2]_{S_0}$, which is started from a specific state S_0 that satisfies $Inv(A_1)$ and $Inv(A_2)$, forms a class of states of the world. If for any state in this class, $Inv(A_1)$ and $Inv(A_2)$ are satisfied, then $Inv(A_1)$ and $Inv(A_2)$ act as state constraints relative to this class of states $R[A_1, A_2]_{S_0}$, and we call $Inv(A_1)$ and $Inv(A_2)$ S_0-*relative state constraints* of the world. Now, let us consider the general case.

Definition 5.2. Let A_1, A_2, $\cdots$, A_n be generic actions of the problem domain, $Inv(A_1)$, $Inv(A_2)$, $\cdots$, $Inv(A_m)$ where $m \leq n$ are invariants of A_1, A_2, $\cdots$, A_m respectively as defined by Definition 8[9], and S_0 a state that satisfies $Inv(A_1)$, $Inv(A_2)$, $\cdots$, $Inv(A_m)$. The set of S_0-relative reachable states of A_1, A_2, $\cdots$, A_n, denoted as $R[A_1, A_2, \cdots, A_n]_{S_0}$, is defined as following:

1. $R_0 = \{S_0\}$,
2. $R_i = R_{i-1} \cup \{S' \mid$ there exists some $S \in R_{i-1}$ and some $act \in A_1 \cup \cdots \cup A_n$ such that $S \models Pre_{act}$ and $S' \in Result(S, act)\}$,
3. $R[A_1, A_2, \cdots, A_n]_{S_0} = \bigcup_{i=0}^{\infty} R_i$.

[9]Note that condition $m \leq n$ is necessary since there may not exist action invariants for some actions as stated previously.

Definition 5.3. Under the same conditions as in Definition 5.2, $Inv(A_1)$, $Inv(A_2)$, $\cdots$, $Inv(A_m)$ are called S_0-*relative state constraints* of the world if for any $S \in R[A_1, A_2, \cdots, A_n]_{S_0}$, S satisfies $Inv(A_1)$, $Inv(A_2)$, $\cdots$, $Inv(A_m)$.

To prove that generalized action invariants are relative state constraints with respect to a class of world states is a domain-dependent task. However, for any given domain where a small number of action descriptions are specified, such proof can be achieved by induction. This is shown by the following example.

Example 5.1. The blocks world (continued). Suppose in the blocks world, there is another kind of action called "pickup". We define the description of action pickup as following:

$$Pre_{Pickup} \equiv block(x) \wedge \forall y.\neg on(y, x) \wedge \exists w.on(x, w), \tag{19}$$

$$Post_{Pickup} \equiv block(x) \wedge \forall z.\neg on(x, z)^{10}. \tag{20}$$

From the above action description, a specific action of picking up block c ($pickup(c)$) is specified as follows:

$$Pre_{pickup(c)} \equiv block(c) \wedge \forall y.\neg on(y, c) \wedge \exists w.on(c, w), \tag{21}$$

$$Post_{pickup(c)} \equiv block(c) \wedge \forall z.\neg on(c, z). \tag{22}$$

Applying the method presented in last section, $P(Persist|pickup(c))_S$ is then expressed by formula $Exp_{pickup(c)} \equiv block(c) \wedge \forall y.\neg on(y, c)$. Simplifying formula $Post_{pickup(c)} \supset Exp_{act}$, we get an invariant of $pickup(c)$:

$$block(c) \wedge \forall z.\neg on(c, z) \supset \forall y.\neg on(y, c), \tag{23}$$

which represents that if c is in hand, then c must be clear.

From formulas (15) — (18) and (23), according to Definition 5.1, we then have the following generalized invariants for action description $Stack$ where all free variables occurring in these formulas are considered to be universally quantified from the outside.

$$on(x, y) \supset block(x), \tag{24}$$

$$\neg(on(x, y) \wedge z \neq y \wedge on(x, z)), \tag{25}$$

$$\neg(on(x, y) \wedge v \neq x \wedge on(v, y)), \tag{26}$$

$$on(x, y) \wedge \neg loc(y) \supset \exists w.on(y, w), \tag{27}$$

and a generalized invariant for $Pickup$:

$$block(x) \wedge \neg on(x, z) \supset \neg on(y, x). \tag{28}$$

□

Now, we can inductively prove the following interesting result.

Theorem 5.1. In the blocks world, where $Stack$ and $Pickup$ are the only two action descriptions, let S_0 be any state that satisfies formulas (24) — (28). Then formulas (24) — (28) are S_0-relative state constraints of the blocks world. That is, for any state $S \in R[Stack, Pickup]_{S_0}$, S satisfies (24) — (28).

[10]Note that the post-condition of $Pickup$ cannot be simply defined as $\forall z.\neg on(x, z)$, since implicitly, the post-condition of $Pickup$ should be $inhand(x)$, and we have the definition $inhand(x) \equiv block(x) \wedge \forall z.\neg on(x, z)$.

Proof:
According to Definition 5.2, $R[Stack, Pickup]_{S_0}$ is:
$R[Stack, Pickup]_{S_0} = R_0 \cup \cdots \cup R_{n-1} \cup R_n \cup \cdots$, where
$R_0 = \{S_0\}$,
$R_n = R_{n-1} \cup \{S' \mid S' \in Result(S, act)$, where $S \in R_{n-1}$,
$act \in Stack \cup Pickup$ and $S \models Pre_{act}\}$.

We prove this theorem by induction on n. Clearly, S_0 satisfies (24) — (28). By inductive hypothesis, any S in R_{n-1} satisfies (24) — (28). Now consider R_n. We need to prove that for each $S' \in R_n$, S' satisfies (24) — (28). According to the above description, for any $S' \in R_n$, $S' \in R_{n-1}$, or there exists some $S \in R_{n-1}$, and some act in $Stack$ or $Pickup$, such that $S \models Pre_{act}$ and $S' \in Result(S, act)$. We only need to consider the latter case. Since (24) — (28) are satisfied in S, we need to prove that performing action act preserves formulas (24) — (28). That is, (24) — (28) are also satisfied in S'. There are two cases, one is $act \in Stack$; the other is $act \in Pick$.

Case 1. Suppose act is any specific action of $Stack$, say $stack(a, b)$, where a, b are object constants.

Since (24) — (28) are satisfied in S, then every instance of (24) — (28) must be true in S. Obviously, the only possibility to make (24), (25), (26), (27), or (28) become false in S' by performing $stack(a, b)$ is that at least one formula, which is obtained from (24), (25), (26), (27), or (28) by replacing those related variables with a, b, becomes false in S'. These formulas are:

$$on(a, b) \supset block(a), \tag{29}$$

$$\neg(on(a, b) \wedge z \neq b \wedge on(a, z)), \tag{30}$$

$$\neg(on(a, b) \wedge v \neq a \wedge on(v, b)), \tag{31}$$

$$on(a, b) \wedge \neg loc(b) \supset \exists w.on(b, w), \tag{32}$$

$$block(a) \wedge \neg on(a, z) \supset \neg on(y, a), \tag{33}$$

$$block(b) \wedge \neg on(b, z) \supset \neg on(y, b). \tag{34}$$

Note that there are two related formulas obtained from (28) by replacing variable x with a, b respectively.

By Example 4.2, we have shown that (29) — (32) are preserved in S' by performing $stack(a, b)$. Now we show (33) and (34) are also preserved in S'. Clearly, from the semantics of $stack(a, b)$, the only difference between S and S' is that in S, $\forall z.\neg on(a, z)$ is true and $\forall v.\neg on(v, b)$ is true, while in S', $on(a, b)$ becomes true. Consider (33). Since $\forall z.\neg on(a, z)$ becomes false in S', then formula $block(a) \wedge \forall z.\neg on(a, z)$ is false in S', this follows that (33) is still true in S'. For (34), on the other hand, since $\forall y.\neg on(b, y)$ is false both in S and S', then formula (34) remains true in S'. Thus, formulas (33) and (34) are preserved by action $stack(a, b)$.

Case 2. Suppose act is any specific action of $Pickup$, say, $pickup(c)$, where c is an object constant. The proof of preserving (24) — (28) in S' is similar to the proof of Case 1. □

6. Conclusions

In this paper, we have proposed a formal method to generate action invariants from their specifications. We also discussed the generalization of action invariants into state constraints. Reasoning about persistence has an important role in this procedure.

As we have mentioned earlier, there are some limitations in our work. First, the expressibility of persistent projections of actions is not completely solved. Theorem 4.4 only covered the case of actions with definite pre-conditions, while Theorem 4.5 provided a partial solution for indefinite cases. Second, to verify if generalized action invariants are *relative* state constraints is generally domain-dependent. This seems to be unavoidable. But our argument is that for any given domain where the number of action descriptions specified is not large, such verification can be achieved by an inductive proof procedure, as shown by the proof of Theorem 5.1.

Although some researchers have noticed that extracting invariants and constraints from action theories has potential applications to dynamic systems modeling, there has not been much systematic work reported in the literature so far. Reiter's approach [10] probably was the first elegant and significant investigation of this issue. Connections between action theories and knowledge/belief systems approach this issue too, but with different motivations (eg., [1]).

Reiter's work and ours presented in this paper are however very different. First, our action theory is based on the state-based persistence formalism, while Reiter's action theory was based on the situation calculus. Second, to derive properties (constraints) from action theories, Reiter introduced induction axioms in the situation calculus, while we achieve this purpose by two steps — first to generate action invariants, and then to generalize invariants to state constraints relative to a class of states reachable by a class of actions. Despite these differences, both Reiter's work and ours address the same problem. The merit of our approach is another way to address the problem, and it insights that the interaction between action specifications and state constraints is obvious to the extent that we may derive the latter from the former using the syntax of logic. The reverse derivation is intimately connected with the update problem, and we are currently investigating it.

References

[1] del Val, A. and Shoham, Y.: "Deriving properties of belief update from theories of action". *Proceedings of Thirteenth International Joint Conference on Artificial Intelligence*, pages 732–737, Morgan Kaufmann Publishers, Inc., Chembary, 1993, 732–737.

[2] Fikes, R. and Nilsson, N.: " STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence*, **Vol** (2), 1971, 189–208.

[3] Ginsberg, M.L. and Smith, D.E.: "Reasoning about action I: A possible worlds approach". *Proceedings of the 1987 Workshop on the Frame Problem in Artificial Intelligence*, Morgan Kaufmann Publisher, Inc., 1987, 233–258.

[4] Lifschitz, V.: "Computing circumscription". *Proceedings of IJCAI-85*, Morgan Kaufmann Publisher, Inc., 1985, 121–127.

[5] McCarthy, J. and Hayes, P.J.: "Some philosophical problems from the standpoint of artificial intelligence". *Machine Intelligence*, **Vol** (4), 1969, 463–502.

[6] Pednault, E.P.D.: "ADL: Exploring the middle ground between STRIPS and the situation calculus". *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, Morgan Kaufmann Publisher, Inc., 1989, 324–332.

[7] Reiter, R.: "Equality and domain closure in first-order databases". *Journal of the Associate for Computing Machinery*, **Vol** (27), 1980, 235–247.

[8] Reiter, R.: "The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression". *Artificial Intelligence and Mathematical Theory of Computation*, V. Lifschitz (ed.), Academic Press, Inc., 1991, 359–380.

[9] Reiter, R.: "On specifying database updates". *KRR-TR-92-3*, University of Toronto, July, 1992.

[10] Reiter, R.: "Proving properties of states in the situation calculus". *Artificial Intelligence*, **Vol** (64), 1993, 337–351.

[11] Schubert, L.K.: "Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions". In H.E. Kyberg, R.P. Loui, and G. N. Carlson, editors, "Knowledge Representation and Defeasible reasoning", H.E. Kyberg, R.P. Loui, and G.N. Carlson (ed.), Kluwer Academic Press, 1990, 23–67.

[12] Winslett, M.: "Reasoning about action using a possible models approach". *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-88)*, Morgan Kaufmann Publisher, Inc., 1988, 89–93.

[13] Zhang, Y. and Foo, N.Y.: "Reasoning about persistence: a theory of actions". *Proceedings of Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, Morgan Kaufmann Publishers, Inc., 1993, 718–723.

[14] Zhang, Y. and Foo, N.Y.: "Applying the persistent set approach in temporal reasoning". *Annals of Mathematics and Artificial Intelligence*, **Vol** (14), 1995, 75–98.

[15] Zhang, Y.: "On propositional knowledge base updates". *Australian Journal of Intelligent Information Processing Systems*, **Vol** (2), 1995, 28–37.